Vorlesungsmitschrift

Semantik von Programmiersprachen

gelesen von Prof. Dr. Elfriede Fehr

Tobias Höppner

SoSe 2014

Inhaltsverzeichnis

1 Einführung (Vorlesung 1 am 17.04.)

1.1 Was sind Programmiersprachen?

Definition 1.1 (Programmiersprachen)**.**
Programmiersprachen sind künstliche, formale Ausdruckssprachen zur Kommunikation zwischen Mensch und Maschine.

Memo technischer Begriff -> z.b. ADD reg1 reg2

Beim Studium von Sprachen unterscheidet man 3 Ebenen(Aspekte):

Syntax einschließlich lexikalischer Struktur (Themen des Übersetzerbaus)
- Kern der Syntax ist die grammatikalische Struktur
- formale Definition durch kontextfreie Grammatiken

Semantik (diese Vorlesung)
- Bedeutung
- Interpretation
  Natürliche Sprachen (Gegenstand der Geisteswissenschaften) lassen Spielräume zur Interpretation offen. Künstliche Sprachen sollen möglichst formalisierbar sein.

Fokus: Formalisierung

Pragmatik Fragen nach dem Gebrauch und Zweck (Useability).
Warum sagt jemand xyz und ist das leicht verständlich?! - Was will jemand damit bewirken?)

1.2 Mehrdeutigkeit in natürlichen Sprachen

Synonyme *Schloss, Schimmel, ...*
Auflösung durch Kontext (meist leicht und unproblematisch)

Satzebene *Dieses Gelände wird zur Verhütung von Straftaten durch die Polizei Videoüberwacht.*
Auflösung durch Hintergrundwissen möglich. Weiteres Beispiel: *Staatsanwaltschaft ermittelt gegen Betrüger in Clownskostüm.*

1.3 Formalisierungsmethoden

In dieser Vorlesung werden drei Formalisierungsmethoden für die Semantik von Programmiersprachen behandelt.

Motivation

- Sicherheit beim Programmentwurf
- Formale Verifikation von Eigenschaften
- Richtlinie Übersetzerbau
- Automatische Erzeugung von Programm aus Spezifikation

Entwicklung der Formalisierungsansätze

operationale Semantik *(Landin 1964)*: Man stützt die Bedeutung auf die Funktionsweise technischer und abstrakte Maschinen ab. Dazu macht man die Maschine so einfach wie möglich und erkläre die Wirkung der Befehle auf die Maschine. Diese Semantik ist ähnlich ähnliche wie die denotationelle Semantik (mathematische Notation), jedoch wirklich näher an der Maschine.

denotationelle Semantik *(McCarthy 1962)*: Formales erfassen durch mathematische Notation. Weitgehende Abstraktion vom Zustandsraum mit einer direkten Zuordnung von syntaktischen Komponenten zu mathematischen Objekten (Semantik).

axiomatische Semantik *(Hoare 1969)*: Veränderung/Transformation von Bedingungen/Prädikaten auf dem Zustandsraum (einer abstrakten Maschine). Das geschieht mit mathematischen Formeln. z.B.: Hoareformel: $\{Q\}P\{R\}$

1.4 Referenzsprache

Um alle drei Formalisierungsmethoden zu betrachten nutzen wir die Referenzsprache **WHILE**.

1.4.1 Definition der Syntax

(Wie ist die Sprache grammatikalisch aufgebaut?!)
Elementare Einheiten

```
// ganze Zahlen (endlicher Ausschnitt der ganzen Zahlen MIN+1 .. MAX)
Z::= 0 |1|...|MAX|-1|-2|...|MIN
// Wahrheitswerte BOOL
W::= TRUE | FALSE
// Konstanten KON
K::= Z | W
// Bezeichner bzw. Variablen mit Indizes
I::= a|b|...|z|a_1|a_2|...|z_i
// Operatoren
OP::= + | - | * | / | mod
// boolesche Operatoren
BOP::= < | > | = | !> | !< | !=
```

Zusätzliche Einheiten *(induktiv)*

```
// Terme TERM
T::= Z  |  I  |  T_1 OP T_2 | READ, für T_1,T_2 in TERM
// boolesche Terme BT
B::= W | T_1 BOP T_2 | read | not B
// Befehle (Zustandstransformation) COM
C::= skip | I := T | C_1; C | if B then C_1 else C_2 | while B do C |
    output T | output B
```

Die Indizes sind dazu da das Vorkommen von Symbolen in der Struktur *eindeutig* zu beschreiben.

Warum braucht man für so eine Sprache eine formale Semantik?!
Ich möchte maschinell arbeiten, aber es gibt Unterspezifikationen, unklar ist das Verhalten bei:

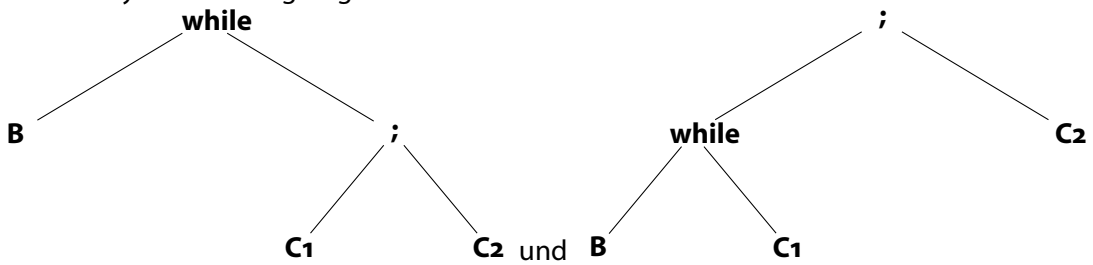
- Typkonflikte
- Fehlerbehandlung
- Rekursion

WHILE ist mehrdeutig?

Ja, das zeigt folgendes Beispiel:

```
while B do C_1; C_2
```

wo beide Syntaxbäume gültig sind.



Um dies zu verhindern werden untergeordnete Befehle eingerückt oder geklammert.

```
while B do
  C1;
  C2
```

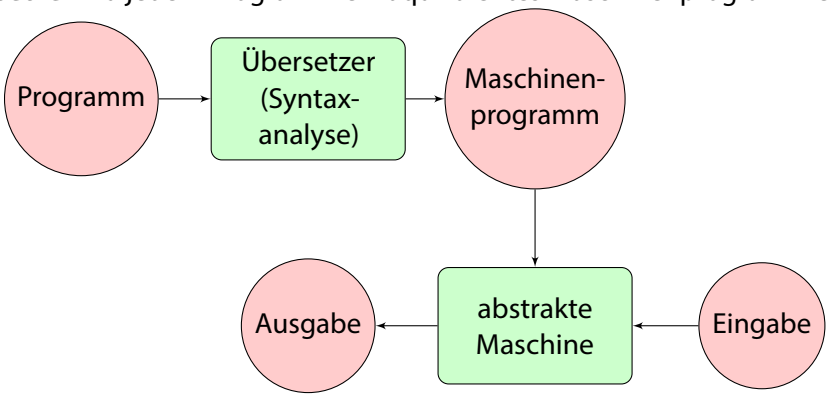
2 Operationelle Semantik (Vorlesung 2 am 24.04.)

2.1 Operationelle Semantik am Beispiel der Terme

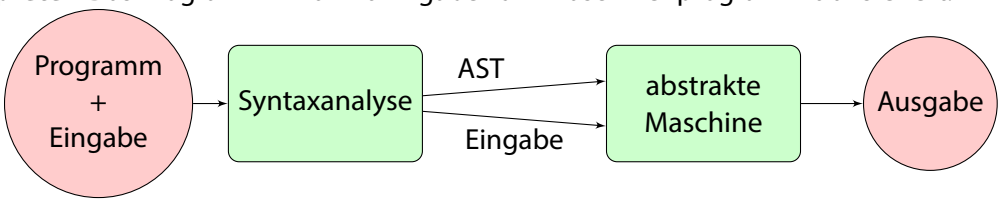
(Inhalt ist nicht im Lehrbuch!)

Es ist wichtig die Struktur von einer Sprache zu kennen, erst dann kann man eine korrekte Interpretation anfertigen!
Grundsätzlich gibt es zwei Methoden:

Übersetzer Zu jedem Programm ein äquivalentes Maschinenprogramm erstellen.



Interpreter Das Programm wird mit Eingabe zum Maschinenprogramm transferiert.



AST: abstract syntax tree

2.2 Terme der Sprache WHILE

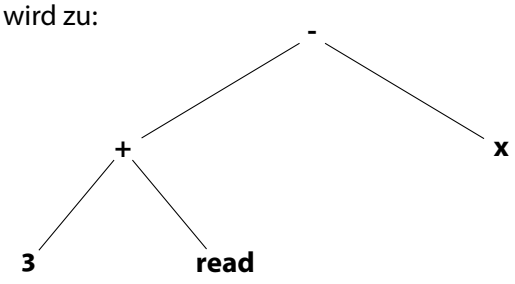
```
// Terme TERM
T::= Z  |  I  |  T1 OP T2 | READ, für T1,T2 in TERM
```

Beispiel: AST

Der Ausdruck

```
3 + read - x
```

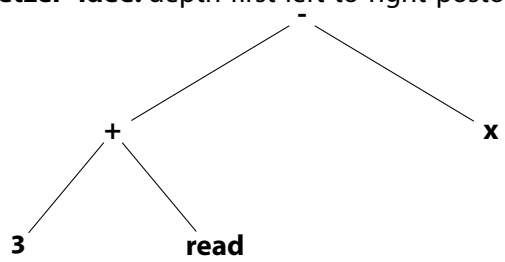
wird zu:



2.3 Informelle Semantik

Interpretation nur möglich, wenn Speicher und Eingabe vorgelegt sind.
Annahme: wir bekommen alles als AST und wir bekommen eine Eingabe, die auch von der Maschine unterstützt wird!

Übersetzer **Idee:** depth-first-left-to-right-postorder Traversierung des AST Unser Beispiel:



wird übersetzt zu:

```
1 PUSH 3
2 READ
3 ADD
4 LOAD x
5 SUB
```

Zustandsveränderungen:
aktueller Zustand (siehe Bild Architektur!):

$$
\begin{aligned}
< \epsilon|S|8.5.... > &\xrightarrow{\texttt{PUSH 3}} && < 3.|S|8.5.... > \\
&\xrightarrow{\texttt{READ}} && < -8.3 - \epsilon|S|5.... > \\
&\xrightarrow{\texttt{ADD}} && < 3 + (-8).\epsilon|S|5.... > \\
&\xrightarrow{\texttt{LOAD x}} && < 2. - 5.\epsilon|S|...5 > \\
&\xrightarrow{\texttt{SUB}} && < -7.\epsilon|S|5... >
\end{aligned}
$$

Semantik eines Terms T zu geg. Speicher S und Eingabe E ist die Spitze des Wertekellers(STACK) nach Ausführung von trans T auf $< \epsilon|S|E >$, falls diese Ausführung fehlerfrei läuft, sonst Fehler!

Interpreter (abstrakte Maschine beinhaltet eine Komponente (Kontrollkeller), in der ASTs in einem Keller gespeichert werden können.)

- Kontrollkeller
- Zustand der abstrakten Maschine hat Komponenten
  - Wertekeller W ($\in ZAHL^*$)
  - Speicher S ($S \in [ID \to ZAHL]$)
  - Kontrollkeller K ($\in (AST \cup OP)^*$)
  - Eingabe E ($\in ZAHL^*$)

Zur Formalisierung der Semantik über die abstrakte Maschine mit dem Zustandsraum Z durch Angabe von:

(i) einem Anfangszustand $Z_{T,S,E}$ für jeden Term T, Speicher S und Eingabe E.
(ii) eine Zustandsüberführungsfunktion $\Delta : Z \to Z$ (partiell)
(iii) Erklärung der Semantik über Iteration von Δ

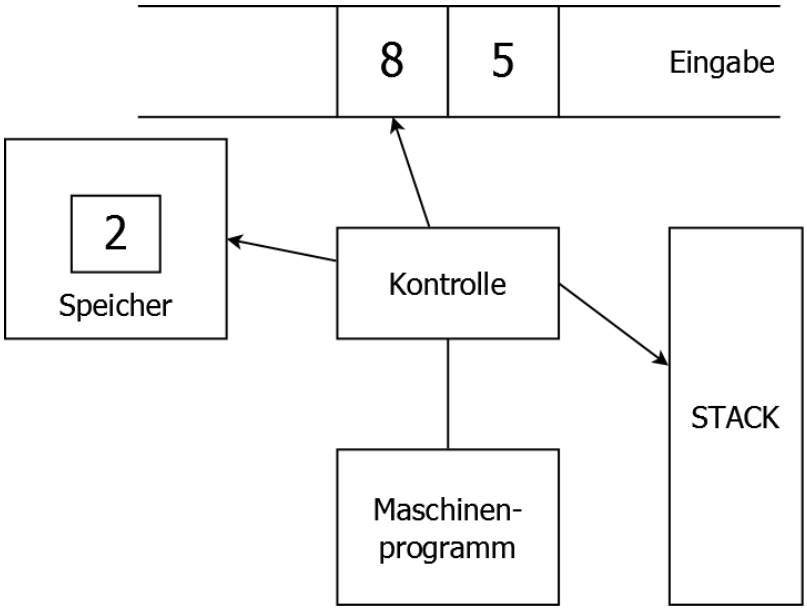
für Terme aus WHILE:

(i) $Z_{T_0,S_0,E_0} :=< \epsilon|S_0|T_0.\epsilon|E_0 >$
(ii) Δ per Induktion über die Struktur der Kontrollkellerspitze

$\Delta < W|S|n.K|E > \quad = < n.W|S|K|E >$ für alle $n \in ZAHL, W, S, E$ wie oben.
$\Delta < W|S|x.K|E > \quad = < s(x).W|S|K|E >$ für alle $x \in ID$
$\Delta < W|S|read.K|n.E > \quad = < n.W|S|K|E >$ für alle $n \in ZAHL$
$\Delta < W|S|T_1 OP T_2.K|E > \quad = < W|S|T_1.T_2.OP.K|E >$
$\Delta < n_2.n_1.W|S|OP.K|E > \quad = < n_1 OP n_2.W|S|K|E > n_1, n_2 \in ZAHL$ falls $n_1 OP n_2$ definiert ist.

(iii) Die Semantik eines (beliebigen) Terms T im Bezug auf einem Speicher S und einer Eingabe E ist $n \in ZAHL$, wenn $\Delta^k Z_{T,S,E} =< n.\epsilon|S|\epsilon|E' >$ für beliebige $E' \in ZAHL^*$, undefiniert sonst!

Architektur der abst. Maschine



Befehlssatz

```
// Stack und Speicher Operationen
READ        // lese Zahl v. Eingabe, PUSH Zahl, rücke Zeiger um eins weiter
LOAD x      // PUSH Inhalt aus Speicher mit symbl. Adr. x
PUSH n      // für jede Zahl aus N lege n auf Stack
(STORE x)   // belegt Speicher mit der symbolischer Adresse x
(GOTO n)    // bedingter Sprung

// arithmetische Operationen
ADD
MULT
SUB
```

3 Operationelle Semantik (Vorlesung 3 am 08.05.)

(Mitschrift von HvB, da Autor im Urlaub. Layout modifiziert von TH.)

3.1 Allgemeines

Methodik einer Formalisierung (Interpreter) der Semantik einer Programmiersprache $\mathcal{P}$
Zur operationellen Semantik gehören insbesondere 3 Angaben:

1. Definition des Zustandsraums einer abstrakten Maschine - möglichst einfach: $\mathcal{Z}$
2. Definition einer (partiellen) Zustandsüberführungsfunktion $\Delta : \mathcal{Z} \to \mathcal{Z}$
3. Definition eines Anfangszustands $\mathcal{Z}_{P,E}$ zu jedem Programm P und Eingabe E

Aus 1.-3. ergibt sich die operationelle Semantik $\mathcal{O}$ von $\mathcal{P}$ wie folgt:
$\mathcal{O} : \mathcal{P} \to [\mathcal{E} \to \mathcal{A} \cup \{Fehler\}]$

$$\mathcal{O}(P)(E) = \begin{cases} A, & \text{falls } \exists k \in \mathbb{N} \text{ mit } \Delta^k(Z_{P,E} = \Delta^{k+1}(Z_{P,E}) \text{ und} \\ & A \text{ die Ausgabekomponente von } Z \text{ ist} \\ \text{Fehler}, & \text{falls es ein } k \in \mathbb{N} \text{ gibt mit } \Delta^k(Z_{P,E}) \text{ nicht definiert} \\ \text{undefiniert}, & \text{sonst.} \end{cases}$$

3.2 Operationelle Semantik von WHILE

1. **Der Zustandsraum $\mathcal{Z}$ ist das kartesische Produkt** $\mathcal{W} \times \mathcal{S} \times \mathcal{K} \times \mathcal{E} \times \mathcal{A}$ mit:

   Wertekeller $W \in \mathcal{W}$ ist eine Folge von Konstanten, d.h.
   $\mathcal{W} = KON^*$
   Speicher $S \in \mathcal{S}$ ist eine Funktion von Bezeichnern nach $ZAHL \cup \{frei\}$, d.h.
   $\mathcal{S} = [ID \to ZAHL]$
   Kontrollkeller $K \in \mathcal{K}$ ist ein Folge von ASTs bzw. Kontrollsymbolen, d.h.
   $\mathcal{K} = \left(TERM \cup BT \cup COM \cup OP \cup BOP \cup \{\underline{if}, \underline{while}, \underline{assign}\}\right)^*$
   Ein- und Ausgabe $E \in \mathcal{E}$ bzw. $A \in \mathcal{A}$ ist jeweils eine Folge von Konstanten, d.h.
   $\mathcal{E} = KON^*$ bzw. $\mathcal{A} = KON^*$

2. **Induktion über den Aufbau der Kontrollkellerspitze**

   a. TERM - $T := Z|I|T_1\ \underline{OP}\ T_2|\underline{read}$
   $\Delta\langle W|S|n.K|E|A\rangle = \langle n.W|S|K|E|A\rangle$ für alle $n \in ZAHL$
   $\Delta\langle W|S|x.K|E|A\rangle = \langle S(x).W|S|K|E|A\langle$ für alle $x \in ID$ mit $S(x) \neq frei$
   $\Delta\langle W|S|\underline{read}.K|n.E|A\rangle = \langle n.W|S|K|E|A\langle$ für alle $n \in ZAHL$
   $\Delta\langle W|S|T_1\underline{OP}\ T_2.K|E|A\rangle = \langle W|S|T_1.T_2.\underline{OP}.K|E|A\rangle$
   $\Delta\langle n_2.n_1.W|S|\underline{OP}.K|E|A\rangle = \langle \underline{(n_1\ \underline{OP}\ n_2)}.W|S|K|E|A\rangle$, falls $\underline{n_1\ \underline{OP}\ n_2}$ nicht aus dem darstellbaren Zahlenbereich herausführt

   b. Boolsche Terme (ähnlich)

   c. COM - $C := \underline{skip}|I := T|C_1; C_2|\underline{if}\ B\ \underline{then}\ C_1\ \underline{else}\ C_2|\underline{while}\ B\ \underline{do}\ C|\underline{output}\ T|\underline{output}\ B$
   $\Delta\langle W|S|\underline{skip}.K|E|A\rangle = \langle W|S|K|E|A\rangle$
   $\Delta\langle W|S|I := T.K|E|A\rangle = \langle W|S|T.\underline{assign}.I.K|E|A\rangle$
   $\Delta\langle n.W|S|\underline{assign}.I.K|E|A\rangle = \langle W|S[n/I]|K|E|A\rangle$, wobei $n \in ZAHL$ und
   $S[n/I](x) = \begin{cases} n, & \text{falls } I = x \\ S(x) & sonst \end{cases}$
   $\Delta\langle W|S|C_1; C_2.K|E|A\rangle = \langle W|S|C_1.C_2.K|E|A\rangle$ $\Delta\langle W|S|\underline{if}\ B\ \underline{then}\ C_1\ \underline{else}\ C_2.K|E|A\rangle = \langle W|S|B.\underline{if}.C_1.C_2.K|E|A\rangle$

$\Delta\langle \underline{true}.W|S|\underline{if}.C_1.C_2.K|E|A\rangle = \langle W|S|C_1.K|E|A|\rangle$
$\Delta\langle \underline{false}.W|S|\underline{if}.C_1.C_2.K|E|A\rangle = \langle W|S|C_2.K|E|A|\rangle$
$\Delta\langle W|S|\underline{while}\ B\ \underline{do}\ C.K|E|A\rangle = \langle W|S|B.\underline{while}.B.C.K|E|A\rangle$
$\Delta\langle \underline{true}.W|S|\underline{while}.B.C.K|E|A\rangle = \langle W|S|C.B.\underline{while}.B.C.K|E|A\rangle$
$\Delta\langle \underline{false}.W|S|\underline{while}.B.C.K|E|A\rangle = \langle W|S|K|E|A\rangle$

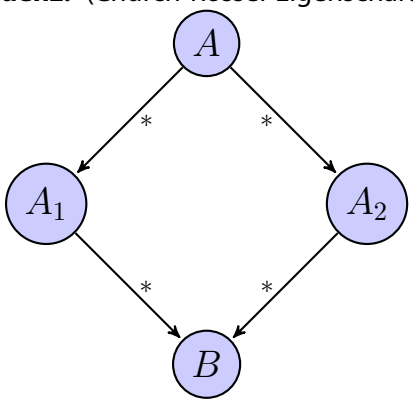
$\Delta\langle W|S|\underline{output}\ T.K|E|A\rangle = \langle W|S|T.\underline{output}.K|E|A\rangle$
$\Delta\langle n.W|S|\underline{output}.K|E|A\rangle = \langle W|S|K|E|n.A\rangle$

4 Reduktionssemantik (Vorlesung 4 am 15.05.)

4.1 Reduktionssemantik

Ausprägungen der operationellen Semantik zu einfacheren Argumentation (Beweisführung) über Programmeigenschaften.
Idee: Reduktion von Ausdrücken, Termen, Programmen und Anweisungen (usw.) auf einfachere aber semantisch äquivalenten Termen (usw.).
Beispiel: Einfache arithmetische Ausdrücke über den natürlichen Zahlen und +, *.
Einzelschrittreduktion: $(4+2)*(7-5) \Rightarrow 6*(7-5) \Rightarrow 6*2 \Rightarrow 12$
allgemeine Form: $A_1 \ \underline{OP} \ A_2 \Rightarrow A_1' \ \underline{OP} \ A_2'$ falls $A_1 \Rightarrow A_1'$ und $A_2 \Rightarrow A_2'$
Axiom: $n_1 \ OP \ n_2 \Rightarrow \underline{n_1 \ OP \ n_2}$
Alternative Schreibweise(Winskel): $\frac{A_1 \Rightarrow A_1', A_2 \Rightarrow A_2'}{A_1 \ \underline{OP} \ A_2 \Rightarrow A_1' \ \underline{OP} \ A_2'}$
Konfluenz: (Church-Rosser Eigenschaft, Diamant)

Church-Rosser gilt nicht bei Nebenwirkungen!



anderes Beispiel: λ-Kalkül. $(\lambda x.M)A \xrightarrow{\beta} \$M$ falls...

4.2 Reduktionssemantik der Sprache WHILE

Zustandsraum: $\mathcal{Z} = \rho \times \mathcal{E} \times \mathcal{A}$ Speicher, Ein- und Ausgabe. $\rho = [ID \to \text{ZAHL} \cup \{\underline{\text{frei}}\}], \mathcal{E} =$ KON*, $\mathcal{A} =$ KON*
$\Rightarrow$ Reduktionsrelation über (TERM $\cup$ BT $\cup$ COM) $\times \mathcal{Z}$
Induktiv über den Aufbau der Syntax:

1. **Terme**: Keine Reduktionsregel für (n, z), d.h. Normalform für $n \in$ ZAHL
   - **a** $(x, (s, e, a)) \Rightarrow (s(x), (s, e, a))$, falls $s(x) \neq \underline{\text{frei}}$ für $x \in ID$, $(s, e, a) \in \mathcal{Z}$
   - **b** $\underline{\text{read}} \Rightarrow (n, (s, e, a)),$ falls $n \in$ ZAHL.
   - **c** $(T_1 \ \underline{OP} \ T_2, z) \Rightarrow (n \ \underline{OP} \ T_2, z')$, falls $(T_1, z) \stackrel{*}{\Rightarrow} (n, z')$
   - **d** $(n \ \underline{OP} \ T_1, z) \Rightarrow (n \ \underline{OP} \ m, z')$, falls $(T, z) \stackrel{*}{\Rightarrow} (m, z')$
   - **e** $(n \ \underline{OP} \ n, z) \Rightarrow (\underline{n \ OP \ m}, z')$, falls $\underline{n \ OP \ m} \in$ ZAHL.
2. **BT** analog.
3. **COM**: keine Reduktionsregel für $\underline{skip}$ (Normalform)
   - **a** $(I := T, (s, e, a)) \Rightarrow (\underline{skip}, (s[n/I], e', a))$, falls $(T, (s, e, a)) \stackrel{*}{\Rightarrow} (n, (s, e', a))$
   - **b** $\underline{output} \ T, (s, e, a) \Rightarrow (\underline{skip}, (s, e', a.n))$, falls $(T, (s, e, a)) \stackrel{*}{\Rightarrow} (n, (s, e', a))$
   - **c** $\underline{output} \ B, (s, e, a) \Rightarrow (\underline{skip}, (s, e', a.b))$, falls $(B, (s, e, a)) \stackrel{*}{\Rightarrow} (b, (s, e', a))$
   - **d** $(C_1; C_2, z) \Rightarrow (C_2, z)$, falls $(C_1, z) \stackrel{*}{\Rightarrow} (\underline{skip}, z')$
   - **e** $(\underline{if} \ B \ \underline{then} \ C_1 \ \underline{else} \ C_2, z) \Rightarrow (C_1, z')$, falls $(B, z) \stackrel{*}{\Rightarrow} (\underline{true}, z')$
   - **f** $(\underline{if} \ B \ \underline{then} \ C_1 \ \underline{else} \ C_2, z) \Rightarrow (C_2, z')$, falls $(B, z) \stackrel{*}{\Rightarrow} (\underline{false}, z')$
   - **g** $(\underline{while} \ B \ \underline{do} \ C, z) \Rightarrow (C; \underline{while} \ B \ \underline{do} \ C, z')$, falls $(B, z) \stackrel{*}{\Rightarrow} (\underline{true}, z')$

h $(\underline{while}\ B\ \underline{do}\ C, z) \Rightarrow (\underline{skip}, z')$, falls $(B, z) \stackrel{*}{\Rightarrow} (\underline{false}, z')$

4.3 Reduktionssemantik (schematisch)

$$\underline{eval}(P)(E) = \begin{cases} A, & \text{falls } (P, (S_0, E, \mathcal{E})) \stackrel{*}{\Rightarrow} (\underline{skip}, (S, E', A)) \text{ mit bel. } S \in \rho \text{ und } E', A \in \text{KON}^* \\ \underline{\text{Fehler}}, & \text{falls } (P, (S_0, E, \mathcal{E})) \stackrel{*}{\Rightarrow} (C, (S, E', A)) \text{ mit bel. } C \in \text{ COM und } E', A \in \text{KON}^* \\ & \text{und } C \neq \underline{skip} \text{ und } (C, (S, E', A)) \text{ lässt sich nicht mehr mit } \Rightarrow \text{ reduzieren} \\ \text{undefiniert}, & \text{sonst.} \end{cases}$$

Satz: $\mathcal{O} = \underline{eval}$ (extensional)
Beweis über strukturelle Induktion.

5 Denotationelle Semantik (Vorlesung 5 am 22.05.)

Grundprinzip:
Direkte (injektive) Zuordnung von syntaktischen Strukturen zu deren Semantik (als mathematisches Objekt (Konstrukt)).

5.1 Allgemeines Prinzip

1. Ordne jeder syntaktischen Kategorie K der Sprache einen semantischen Bereich B_K zu.
2. Schreibe (definiere) zu jeder syntaktischen Kategorie eine Semantikfunktion $\mathcal{K}$
   $\mathcal{K} : K \to B_K$

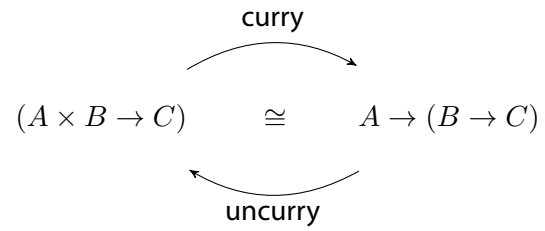
Notation: *(Klammern sparen!)*
- Das Argument einer Semantikfunktion $\mathcal{K}$ schreiben wir in $[\![$ und $]\!]$
- Funktionen z.B. $f : A \to (B \to C)$ werden $f\ a\ b = a \ldots b \ldots$

5.2 Denotationelle Semantik

- **Syntaktische Kategorien** von WHILE: TERM, BT, COM, PROG
- Hilfsbereiche:
  ZUSTAND = SPEICHER $\times$ EINGABE $\times$ AUSGABE mit SPEICHER = ID $\to$ ZAHL $\cup\{\underline{frei}\}$,
  EINGABE = KON* und AUSGABE = KON* *(Abh. vom Speicher + Nebenwirkung)*

1. $\mathcal{T}$:TERM$\to \underline{\underline{B_T}}$
   B_T :=ZAHL???

Exkurs: **Curry-Isomorphie**



curry: $f\ a\ b = f(a, b)$
uncurry: $g\ (a, b) = g\ a\ b$
$B_T :=$ SPEICHER $\to$ EINGABE $\to$ ((ZAHL $\times$ Eingabe) $\cup\{\underline{Fehler}\}$)
$\mathcal{B}$: BT $\to$ SPEICHER $\to$ EINGABE $\to$ ((BOOL $\times$ Eingabe) $\cup\{\underline{Fehler}\}$)
$\mathcal{C}$: COM $\to \underbrace{\text{ZUSTAND} \to \text{ZUSTAND} \cup \{\underline{Fehler}\}}_{B_C}$)

2. Induktive Definition über dem Aufbau

$$\mathcal{T}[\![n]\!]\ s\ e = (n, e) \quad \text{für alle } n \in \text{ZAHL}$$

$$\mathcal{T}[\![x]\!]\ s\ e = \begin{cases} (n, e), & \text{falls } s\ x = n \\ \underline{\text{Fehler}}, & \text{falls } s\ x = \underline{frei} \text{ für alle } x \in \text{ID} \end{cases}$$

$$\mathcal{T}[\![\underline{read}]\!]\ s\ e = \begin{cases} (n, e'), & \text{falls } e = n.e' \\ \underline{\text{Fehler}}, & \text{falls } e = \mathcal{E} \text{ oder } e = b.e' \text{ mit } b \in \text{BOOL} \end{cases}$$

$$\mathcal{T}[\![\underline{T_1\ OP\ T_2}]\!]\ s\ e = \begin{cases} (n, e''), & \text{falls} \quad \mathcal{T}[\![T_1]\!]se = (n_1, e') \text{ und} \\ & \qquad \mathcal{T}[\![T_2]\!]se' = (n_2, e'') \text{ und} \\ & \qquad \underline{n_1\ OP\ n_2} = n \in \text{ZAHL} \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{B}[\![w]\!]\ s\ e = (w, e) \quad \text{für alle} w \in \text{BOOL}$$

$$\mathcal{B}[\![\underline{not} B]\!]\ s\ e = \begin{cases} (b, e'), & \text{falls } \mathcal{B}[\![B]\!]se = (w, e') \text{ und } b = \neg w \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{B}[\![T_1\ \underline{BOP}\ T_2]\!]\ s\ e = \begin{cases} (b, e''), & \text{falls} \quad \mathcal{T}[\![T_1]\!]se = (n_1, e') \text{ und} \\ & \qquad \mathcal{T}[\![T_2]\!]se' = (n_2, e'') \text{ und} \\ & \qquad \underline{n_1\ BOP\ n_2} = b \in \text{BOOL} \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{B}[\![\underline{read}]\!]\ s\ e = \begin{cases} (b, e'), & \text{falls } e = b.e' \text{ mit } b \in \text{BOOL} \\ \underline{\text{Fehler}}, & \text{falls } e = \mathcal{E} \text{ oder } e = n.e' \text{ mit } n \in \text{ZAHL} \end{cases}$$

alt. Schreibweise:
$\mathcal{C}[\![\underline{skip}]\!]\ z = \underline{id}$
$z = (s, e, a)$

$$\mathcal{C}[\![\underline{skip}]\!]\ z = z$$

$$\mathcal{C}[\![I := T]\!]\ (s, e, a) = \begin{cases} (S[n/I], e', a), & \text{falls } \mathcal{T}[\![T]\!]\ s\ e = (n, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{C}[\![\underline{output}\ T]\!]\ (s, e, a) = \begin{cases} (s, e', a.n), & \text{falls } \mathcal{T}[\![T]\!]se = (n, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{C}[\![\underline{output}\ B]\!]\ (s, e, a) = \begin{cases} (s, e', a.b), & \text{falls } \mathcal{B}[\![B]\!]se = (b, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{C}[\![C_1; C_2]\!]\ z = \begin{cases} \mathcal{C}[\![C_2]\!]z', \mathcal{C}[\![C_1]\!]z = z' & \text{mit } z' \in \text{ZUSTAND} \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{C}[\![\underline{if}\ B \underline{then}\ C_1\ \underline{else}\ C_2]\!]\ (s, e, a) = \begin{cases} \mathcal{C}[\![C_1]\!]\ (s, e', a), & \text{falls} \quad \mathcal{B}[\![B]\!]\ s\ e = (\underline{true}, e') \\ \mathcal{C}[\![C_2]\!]\ (s, e', a), & \text{falls} \quad \mathcal{B}[\![B]\!]\ s\ e = (\underline{false}, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{C}[\![\underline{while}\ B\ \underline{do}\ C]\!]\ (s, e, a) = \begin{cases} \mathcal{C}[\![C; \underline{while}\ B\ \underline{do}\ C]\!](s, e', a), & \text{falls} \quad \mathcal{B}[\![B]\!]\ s\ e = (\underline{true}, e') \\ (s, e', a), & \text{falls} \quad \mathcal{B}[\![B]\!]\ s\ e = (\underline{false}, e') \\ \underline{\text{Fehler}}, & \text{sonst} \end{cases}$$

$$\mathcal{P}[\![P]\!]\ e = \begin{cases} a, & \text{falls} \quad \mathcal{C}[\![C]\!](s_0, e, \epsilon) = (s, e', a) \\ \text{Fehler}, & \text{falls} \quad \mathcal{C}[\![C]\!](s_0, e, \epsilon) = \underline{Fehler} \\ \text{undefiniert} & \text{sonst} \end{cases}$$

6 Axiomatische Semantik (Vorlesung 6 am 05.06.)

Am Beispiel von WHILE' ($\underline{read}\ I$ anstelle von Termen der Form $\underline{read}$).

Grundlage: C. A. R. Hoare: *An aximatic Basis for Computer Programming* (CACM 1969)

(Hoare-Kalkül zum Bew. von Prog. Eigenschft.)

6.1 Allgemeine Methode

Annahme: AST ist gegeben.

Neben der abstrakten Syntax brauchen wir vier Angaben:

(formales Setting von logischen Ausdrücken vorgeben, damit man damit arbeiten kann!)

1. Eine Menge von Bedingungen (*logische Ausdrücke, Prädikate, assertions*), die über den Zustandsraum interpretierbar sind. Im Allgemeinen Prädikatenlogik 1. Stufe oder Aussagenlogik.
   (formales Setting von logischen Ausdrücken vorgeben, damit man damit arbeiten kann!)
   Beispiel: $x < y, input = \epsilon, \exists i.x = p.i, ...$ natürliche Interpretation: Verbinde $I \in$ ID mit $S(I)$ sowie $\underline{input}$ mit Eingabe....
   Sei $S(x) = 0$, dann ist die Bedingung $y/x = 7$ falsch.

2. Definition zu jeder atomaren Anweisung C ein Axiom bzw. ein Axiomenschema der Form $\{Q\}C\{R\}$ - Hoare Formel
   zu lesen als: Wenn die Bedingung Q auf einem Zustand z gilt und die Ausführung von C auf z in einem Zustand z' terminiert, dann gilt R auf z'.
   Beispiel: für $\underline{skip}$ des Axiomenschema: $\{Q\}\ \underline{skip}\ \{Q\}$, *zu lesen: für alle Q gilt es*.

3. Für zusammengesetzte Anweisungen C mit unmittelbaren Komponenten $C_1, \ldots, C_r$ eine Schlussregel der Form $\frac{F_1,\ldots,F_i}{\{Q\}C\{R\}}$, wobei die F_i Hoare-Formeln zu $C1, \ldots, C_r$ oder andere Formeln sind.

4. *Logischer Klebstoff*: Allgemeine Schlussregeln in Verbindung mit Hoare-Formeln, mindestens (oft ausreichend) die Konsequenzregel: $\frac{Q \Rightarrow S, T \Rightarrow R, \{S\}C\{T\}}{\{Q\}C\{R\}}$, wobei $\Rightarrow$ logische Herleitung bezeichnet.

6.2 Konkretisierung am Beispiel von WHILE'

$T ::= Z \mid I \mid T_1\ \underline{OP}\ T_2$

$B ::= \underline{true} \mid \underline{false} \mid \underline{eof} \mid T_1\ \underline{Relop}\ T_2$

$C ::= \underline{skip} \mid \underline{read}\ I \mid I := T \mid \underline{output}\ T \mid C_1; C_2 \mid \underline{if}\ B\ \underline{then}\ C_1\ \underline{else}\ C_2 \mid \underline{while}\ B\ \underline{do}\ C$

1. Aussagenlogik:
   - Terme über $Z, I, OP, \underline{input}, \underline{output}$, Konstruktor- und Selektorfunktionen (., $\underline{hd}, \underline{tl}$)
   - Bedingungen: $T_1\ \underline{Relop}\ T_2$
     $B_1 \vee B_2, B_1 \wedge B_2, \neg B$

2. atomare Definitionen:

$$\{Q\}\underline{skip}\{Q\} \tag{A.1}$$

(zu lesen: Für jede Bedingung Q gilt die Hoare-Formel $\{Q\}\underline{skip}\{Q\}$)

$$\{Q[T/I]\}I := T\{Q\},$$

wobei$[T/I]$ die einmalige textliche Substitution von T für jedes Vorkommen von I in Q bezeichnet.

(A.2)

$$\{Q[\underline{hd}\ \underline{input}/I \mid \underline{tl}input/\underline{input}]\ \underline{read}\ I\{Q\} \approx I := \underline{hd}(\underline{input}); \underline{input} := \underline{tl}(\underline{input}) \tag{A.3}$$

$$\{Q[output.T/\underline{outpunt}]\}\ \underline{output}\ T\{Q\} \approx \underline{output} = \underline{output}.T \tag{A.4}$$

Beispiele:

$$\{input = (4,5) \wedge 2 + x = 3\}I := 2 + x\{input = (4,5) \wedge I = 3\}$$

$$\{\underline{hd}\ \underline{input} = 7 \wedge \underline{tltl}\ \underline{input} = (3)\}\underline{read}x\{x = 7 \wedge input = (3)\}$$

$$\{\underline{output}.7 + y = (1,2,3,4,5)\}\underline{output}(7+9)\{\underline{output} = (1,2,3,4,5)\}$$

3.

$$\frac{\{Q\}C_1\{S\}, \{S\}C_2\{R\}}{\{Q\}C_1, C_2\{R\}} \tag{A.5}$$

$$\frac{\{Q \wedge B\}C_1\{R\}, \{Q \wedge \neg B\}C_2\{R\}}{\{Q\}\underline{if}\ B\ \underline{then}\ C_1\ \underline{else}\ C_2\{R\}} \tag{A.6}$$

$$\frac{\{Q \wedge B\}C\{Q\}}{\{Q\}\underline{while}\ B\ \underline{do}\ C\{Q \wedge \neg B\}} \tag{A.7}$$

4. *(nicht in Vorlesung behandelt, da wohl nicht so wichtig.)*